Text Mining Exercise, Monday, December 9

Kasper Jensen (kasjens@cbs.dtu.dk)

Introduction

Imagine having a research question and no resource or database of any kind to help you. This is a common issue in cutting-edge science.

Text mining is a tool to help you compile your own resource from raw text sources such as patient records or PubMed abstracts (http://www.ncbi.nlm.nih.gov/pubmed).

In particular, natural language processing has become popular because the method allows for rapid, automated extracting of meaningful information from natural language. In natural language processing a machine is trained to mimic the human text perception.
In the following exercise you will be introduced to basic principles of natural language processing (bag-of-words) and text-mining through a hands-on exercise.

Logon to the server

Do the text-mining exercise on the CBS server. If you are connected to the Internet through wireless you should first logon to the CBS network using your terminal.

NB! If you are on a Windows machine you do not need thinlinc. Instead you should download and use PuTTY as terminal:
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
Follow the step-by-step guide in the supplementary material.

ssh studXXX@padawan.cbs.dtu.dk
XXX is the number given to you during class.

Setup your working folder

First we will make a workspace for this exercise, for you to work in. Setup a workspace for the text-mining exercise using the following commands.

mkdir textmining
cd textmining

Create a link to the exercise files from your workspace by typing.

ln -s /home/projects/kasjens/textmining/textmining_exercise/* .
Remember to include the "." at the end of the sentence when typing.

Your dictionaries

Dictionaries define the words, which will eventually form your resource. The dictionaries contain the words that are to be recognized by the language processor and are one of the most critical input components to your resource. A good dictionary usually takes months to construct but for demonstration purposes we have created a dictionary set for you. In the folder *dataset* in your workspace you will find the dictionary set.
The purpose of the dictionary set is to extract relationships between the consumption of plants and plant-foods and with occurrence of human disease phenotypes.
The dictionary files can be found in the folder *dataset*. Have a look at the content of dictionary files by typing:

head dataset/plant_dictionary.tsv
head dataset/disease_dictionary.tsv

Question 1) What are the 5 first entities in each of the dictionary files?

Tag your documents

The raw text documents from which the information is to be extracted are first "tagged" with the words in dictionary. During tagging, the occurrence of the dictionary words and their position is determined and written to a text file.

The tagging of words in the raw text documents is a critical step in text mining and there are a variety of methods applying techniques ranging from rule based tagging to fuzzy logics and part-of-speech recognition.
You can find an explanation of fuzzy matching at the wiki-page:
http://en.wikipedia.org/wiki/Fuzzy_matching
Or an explanation of part-of-speech tagging at the wiki-page:
http://en.wikipedia.org/wiki/Part-of-speech_tagging

In this exercise we will work with a simple rule based tagger. The following command will tag the abstracts of a shorted version of PubMed (http://www.ncbi.nlm.nih.gov/pubmed) using the words from your dictionaries.

cat dataset/short_medline13.tsv | ./tagger.py --process 1 dataset/noisy_dictionary_4_2_2013.db > tagged_documents.tsv &
NB! If you are trying to copy and paste the command into your terminal you have to remove the newline character.

The command ‘cat’ reads the abstracts from the file ‘short_medline13.tsv’ and ‘|’ pipes the text to a program (python) which will use ‘1’ CPU to tag the documents.
The file ‘dataset/ noisy_dictionary_4_2_2013.db’ is a python shelve compiled from your dictionaries.
The python shelve is compiled from your dictionaries to allow for high-performance computing. However, you do not need to think too much about this at the time being.
The command ‘>’ directs the output of the tagger to a file called ‘tagged_documents.tsv’ and the ‘&’ tells the system to run the job in the background.

The tagging will take approximately 5-6 minutes. You can follow the process by typing:
tail -f tagged_documents.tsv
Exit by ctrl+c

While the tagging is running, spend some time browsing through your file by typing:
less tagged_documents.tsv
Exit by ctrl+c

Question 2) Are there any words that you think occurs more frequently then others? If so, mention one or two of these words.

Blacklisting words which are too common

In this part of the exercise you are to blacklist words that are too common to be of interest. Words that are tagged too frequent may be too common to be of interest.

An example:
The word ‘syndrome’ is a valid disease term. However, the term is not very specific and do not specify anything biological meaningful. Therefore, the word should be put on the blacklist. The same applies to the word ‘isolated’, which is a synonym for the inflammatory disease of the myocardium called ‘isolated fielder’s’. However this term is also too common to be of interest and should also be put on the blacklist.

It may be difficult to decide whether a term should be included or not. However, the overall goal is to remove words that produce too much noise, when in the following step will be training the language processor.

Start going through the tagging frequencies by typing:

./wash_by_blacklist.py tagged_documents.tsv

Question 3) What are the 5 most common terms to be excluded?

The above program it will produce a file called ‘blacklist.words’ that contain expressions that the tagger will use to cleanup your dictionary. It would take days to go through all the terms in detail so we have prepared a ‘cleaned’ dictionary for you to work with. Do not worry about going through the whole list.

Training the language processor

In the following we will train a classifier using the “bag of words” principle to combine the tagged words into meaningful pairs.
Several methods for this exist. In the following we are going to train a naïve Bayes classifier as a language processor (text classifier). The Bayes classifier is a simple probabilistic classifier, based on Bayes' theorem with strong (naive) independence assumptions.

First we need to produce a new tagging using the cleaned dictionary.

cat dataset/short_medline13.tsv | ./tagger.py --process 1 dataset/fixed_dictionary_4_2_2013.db > tagged_documents.tsv &

Use the ‘tail’ command to be sure that the tagging has completed.
tail -f tagged_documents.tsv
Exit by ctrl+c

Now we will train a language processor to link your pairs. Start the program by typing:
./compile_training.py tagged_documents.tsv

Look at the words marked with red and decide whether the text mentions a 'positive', a 'negative' or 'no' correlation between plants and diseases. The tagged words are highlighted in red.

During the training you are likely to find a lot of meaningless pairs. Do the training until you have a couple positives/negatives examples. For an acceptable performance you would need about 50 positive/negative examples. To make the training easier for you the program will handle positive and negative pairs the same.

The program creates a file called ‘training.tsv’. Use the command ‘less’ to inspect the file.
less training.tsv
Exit by ctrl+c

Question 4) What is the structure/content of the file?

Evaluating the performance

The performance is evaluated by classifying known abstracts into *true* and *false* categories. If the abstract is more likely to mention a true relation it is considered true, otherwise false. The method uses 10-fold cross-validation to avoid over-fitting.
http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29

Evaluating the performance is a critical step and several evaluations and tests may be needed to ensure a high performance model. For the evaluation there are two important things to notice.

1. If your training set is invalid the classifier will never reach a high performance.
2. If your training set is too small, the performance of the classifier becomes unstable.

To evaluate the performance of your training, type the following:

./evaluate_performance.py training.tsv > evaluation.tsv &

In case you are not able to compile a good training set in time you can also try the set that I have compiled for you.

./evaluate_performance.py admin/drug_metabolite_training.tsv > evaluation.tsv &

This will produce a file 'evaluation.tsv' and will take 1-2 minutes.

When the evaluation has finished have a look at the accuracy by typing:
gnuplot -e "filename='evaluation.tsv'" plot_evaluation.plg

Question 5) What do you see from the graph? Do you find that the performance of your language classifier stabilizes? If so, how many word features are required?

The evaluation produced a file called 'evaluation.tsv'. Have a look at this file by typing:
less evaluation.tsv
Exit by ctrl+c

Question 6) The first column is the number of features, the second is the accuracy and the last column is the word feature. Do you think there are some words that contribute more to performance then others?

Question 7) The word features are selected using the term-frequency-inverse document frequency (http://en.wikipedia.org/wiki/Tf–idf). If the frequency in which a term occurs in a document is $tf(t,d) = \frac{d}{D}$ and the inverse document frequency (background noise) is $idf = \log\left(\frac{D}{1+d}\right)$, where D is the total number of

documents and d is the number of documents where the term occurs. What is the td-idf score for the following terms?

Term	**d**	**D**	**td-idf score**
and	99	100	
treatment	50	100	
addition	1	100	